THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 4, April 1982
Printed in U.S.A.

No. 10A Remote Switching System:

Remote Terminal Firmware

By D. A. ANDERSON, D. R. FULLER, D. R. HANSON, and R. B. SCHMIDT

(Manuscript received April 29, 1980)

The No. 10A RSS remote terminal (RT) is a part of a distributed telephone switching system consisting of one or more remote terminals controlled by a host ESS via data link communication. The RT is a microprocessor-based peripheral controller acting as an extension of the host network. The RT firmware contains a process-oriented operating system. Ringing and other high-power signals are supplied by universal service circuits used in a time-sharing arrangement. The RT firmware is responsible for scanning lines and reporting supervisory state changes to the host. These changes are reported over either the data link or a channel, depending on the state of the line. If data link communication fails, the RT leaves the normal master/slave mode and enters a special stand-alone mode to provide a POTS-type intra-RT service.

I. INTRODUCTION

The No. 10A RSS RT acts as an intelligent peripheral controller which follows call processing and maintenance orders sent over the data link from the host processor. In addition, the RT performs a large number of autonomous maintenance activities to ensure that all of its functions are being done correctly. In the event that data link communication with the host is lost, the RT is also designed to enter a stand-alone mode that provides simple intra-RT call service to its connected customers.

As with other telephone switching systems, high availability is also a requirement in the RT. This is achieved in part by duplication of the data link and duplication of the microprocessor controller (processor,

memory, peripheral access). The normal state is for one data link to be active and the other standby and for one microprocessor to be active and the other standby. Normally, the standby microprocessor is in a halted state that allows the active microprocessor to write simultaneously into both the active and standby memories. This keeps the standby memory up-to-date in the event that it is necessary to switch microprocessors. A certain portion of the RT firmware is devoted to the management of these duplicated units to ensure that the RT is in a working configuration and is capable of recovering from a hardware fault in one of these units.

In the development of the RT firmware, several program design techniques were used in an attempt to produce a more understandable, maintainable, and easily debugged code. These techniques included peer reviews at the program requirement, design, and code levels, extensive use of the high-level language C, and modular hierarchical program structures. Modules were coded as C functions or as assembly language subroutines using the same linkage conventions as the C compiler. Module coding standards were very important and helped the many programmers involved produce a uniformly good and understandable product. Coding standards required a comment prologue at the beginning of each module describing its purpose and inputs and outputs. Also important were the use of structured control constructs, with proper indentation, and the elimination of unnecessary "go-to's." Module sizes were normally kept to a couple of pages of program listing for understandability.

II. OPERATING SYSTEM STRUCTURE

The techniques described above are sufficient for programming a single task well. However, since in the RT many tasks must execute concurrently, the concept of an operating system becomes attractive to solve the complex problems that arise in a multitask environment. Also the local data hiding aspects of processes and the global data hiding aspects of monitors provide a good structure to limit and control interactions between tasks.

A process has a single entry point, which is called by the operating system, and returns when it is done. A process reads and writes only its own local (private) data and communicates with other processes through monitors. A monitor reads and writes global (shared) data and consists of the set of routines allowed to read or write a particular global data area. A monitor routine is passive in the sense that it only executes when called by a process. Monitors are necessary when two or more processes must communicate and perform the function of hiding the global data structure where the communication actually

takes place. Monitors are especially useful in the RT to hide the data structures necessary for seizing and releasing peripheral resources.

2.1 Scheduling mechanisms

The operating system supports three general categories of program activity: (*i*) base level, (*ii*) interrupt level, and (*iii*) reset level. Most processes execute at base level under control of the base-level scheduler. The operating system is nonpreemptive in that processes give up control voluntarily. The interrupt and reset levels are entered only as the result of a hardware interrupt. Interrupts always return to the point of interruption, whereas resets, which are caused by error conditions, may not return to the point of interruption. Each interrupt enters a separate program structure with its own stack and is overseen by a handler that performs the simple process scheduling required for that level.

The base level scheduler is actually a hierarchy consisting of a main scheduler and several subschedulers. The main scheduler provides two general forms of priority: (*i*) classed priority and (*ii*) timed priority, where priority simply means the frequency at which a process receives control. Under classed priority a process may be in one of three classes termed A, B, and C. Class A processes receive control twice as often as class B and four times as often as class C, according to the sequence ABACABA which is repeated endlessly. Under timed priority, a process receives control at fixed time intervals with a minimum period of once every 100 ms to a maximum period of once every 24 hours. The fixed time interval must be a multiple of 100 ms, 1 second, 1 minute, or 1 hour.

The class C-to-class C time interval is approximately 50 ms under no load and increases as the load increases. Timed scheduling is performed as required between the scheduling of each class with a maximum rate of once per class (or equivalently, 8 times per C-to-C cycle). When the C-to-C cycle time exceeds 800 ms, the system is in overload and some random 100-ms work cycles will be skipped. Each process when called gives up control voluntarily and is designed to keep most of its time segments under 10 ms. As protection against insane processes, a hardware sanity timer causes a reset if not restarted frequently by the operating system.

The base level scheduling algorithm has been kept fairly simple to minimize the overhead consumed by this portion of the operating system. Basic scheduling information is kept in a set of vector tables defining entry points and a set of cycle tables defining frequency of entry. These tables are built at compile time and, thus, define permanent scheduling information. This table-driven approach has the advantage of being easy to administer whenever changes are made.

Interrupt level is used to provide control for those processes that have critical timing requirements. The two normally occurring interrupts are the active data link interrupt and the 10-ms interrupt. A data link interrupt occurs each time the data link hardware is ready to transmit or has received a byte of information. Received bytes are buffered at interrupt level for later examination at base level where complete data link input messages are routed to the appropriate clients by the operating system. Data link output messages are also loaded into the active transmit buffer by the operating system at base level and sent out byte by byte at interrupt level. Of lower priority than the data link interrupt is the 10-ms interrupt which is used to control several processes that require timing more accurate than available at base level. These processes are associated with ringing telephones and surpervising lines at a fast rate.

A reset is the highest priority interrupt and in some cases is not maskable. A reset may be caused manually but in general it is caused by a check circuit sensing an error condition, such as a sanity timer timeout, parity error, write-protect violation, etc. Many of the recovery routines execute only in the event of a reset since they perform functions associated with switching microprocessors and system initialization. As in other program-controlled switching systems, the RT firmware provides several forms of initialization less severe than a complete initialization. Repeatedly occurring resets cause an automatic escalation in the severity of initialization performed, up to the point of clearing all transient telephone calls. Normally, a complete initialization can only be induced manually.

2.2 Memory management

The memory management needs of the RT are influenced by the architecture of the microprocessor used, namely, the *BELLMAC**-8. Assembly language instructions of the *BELLMAC*-8 refer to sixteen 16-bit general registers, but these registers are actually located in memory at a point determined by the address contained in a 16-bit hardware register called the rp (register pointer). The rp can be loaded with any memory address (on an 8-byte boundary) and generally points to an area called the rp stack, since the contents of the general registers can be saved and restored quickly by proper bumping and debumping of the rp. There is also a 16-bit hardware register called the sp (stack pointer) which points to an area called the sp stack. Return addresses, register contents, and other variables can be pushed, popped, or otherwise accessed on this stack. By the appropriate saving and restoring of these two hardware pointers, the context switching

* Trademark of Western Electric.

required when calling subroutines, handling interrupts, or switching processes can be made very efficient in the *BELLMAC*-8.

The *BELLMAC*-8 has 16-bit addressing capability which allows the system to access 64K bytes (where K is 1024). To control peripheral hardware, the RT uses a memory-mapped I/O scheme. The first 32K of the address spectrum addresses memory and the last 32K addresses peripheral hardware. Since 32K of memory is not nearly enough for the RT firmware, a memory-bank switching scheme has been implemented. In the same address spectrum as the periphery, several 32K banks of memory are placed. (See Fig. 1.) A bank select register, external to the *BELLMAC*-8 and under operating system control, decides which bank is selected.

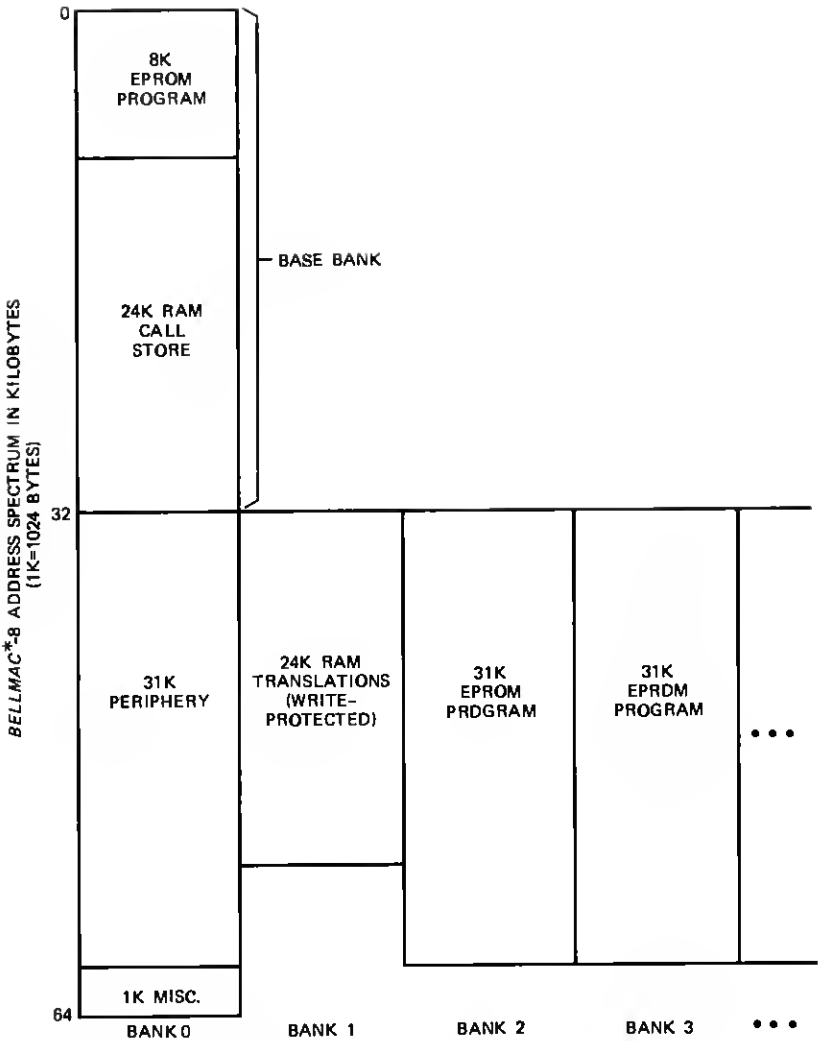
The first 32K or base page of memory is always addressable. All stacks and other writable variables are located here to be accessible to programs executing in any bank. Interrupt handlers, programs that modify the bank select register, and programs that physically read and write the periphery must also be located in the first 32K.

It is important to hide from programmers the linkage required when a subroutine in one bank calls a subroutine in another bank. This linkage is provided automatically for RT firmware by a combination of an overlay link-editor and the trap handler of the operating system. When the link-editor encounters an interbank call, it replaces the call instruction with a special illegal instruction and a pointer to a transfer table entry. The transfer table built by the link-editor contains subroutine addresses for interbank calls, plus the corresponding bank numbers. When the *BELLMAC*-8 executes an illegal instruction, a reset-like interrupt occurs to the trap handler located in the first 32K of memory. If the trap is caused by the link-editor, the trap handler stacks the current return address and bank number, loads the bank number of the called routine, and makes the actual call. Upon return from the called routine, the trap handler restores the previous return address and bank and returns to the calling routine. Traps not caused by the link-editor are treated like an error reset.

Since interbank calls involve more overhead than intrabank calls, the placement of routines in banks is not done at random. Instead, an attempt is made to place related routines in the same bank. The placement policy of routines in banks is administered manually and achieves a best-guess form of optimization.

2.3 Process management

Before describing the implementation of process management in more detail, it is necessary to define the term process descriptor. A process descriptor is the block of writable memory associated with a process that defines the current state of the process. The process



*TRADEMARK OF WESTERN ELECTRIC

Fig. 1—Memory bank structure.

descriptor remains unchanged, while the process takes a time break and for this reason the contents of the process descriptor is very important in determining the program structure of the process. For example, a process descriptor containing a stack allows the process to take a time break while nested several subroutine levels deep. This feature can be extremely valuable for producing an understandable hierarchical program structure.

However, process descriptors with stacks also raise some important questions. How big should the stacks be? Should stacks be dynamically allocated to conserve memory? Can enough concurrent processes be provided to meet system requirements? Is this a reasonable way to utilize memory? The remainder of this section will attempt to explain how these questions were answered for the RT firmware.

First, telephone traffic is well understood and it is possible to estimate the arrival rates and holding times of the various tasks required of the RT. Second, the stack usage conventions of the C compiler are known, and it is possible to estimate stack sizes for storing local variables and passing parameters. The results of this brief examination indicated that it would be very expensive to support each task to be done with the same general-purpose process descriptor.

The RT solution to this general problem was to break the system into several different kinds of processes and to provide a different kind of process descriptor for each. Large descriptors with more general layouts were provided for tasks with few concurrent processes and smaller descriptors with more specific layouts for tasks with many concurrent or real-time intensive processes. This approach tends to make the operating system special purpose, but it appeared to be the only way of incorporating the process concept into the RT firmware and, yet, efficiently meet the system requirements.

The seen and unforeseen complications of dynamic stack allocation were eliminated by taking a conservative approach and not providing this feature. Instead, the size and number of process descriptors is fixed at compile time. The only dynamic stack allocation done at execution time is that which occurs for the allocation of local variables within functions when they are called. Interrupt handlers also have their own dedicated stack areas so that room does not have to be provided for interrupt variables in each individual base-level stack area.

Stack sizes are an important consideration and were determined by the following methods. Initially, estimates were made based on providing a certain number of subroutine nesting levels. Later, a software tool was developed to statically examine written code and chart the nesting structure of processes to determine their deepest stack usages. Stack overflow violations were corrected by changing the code or enlarging the stack. Finally, a stack audit was incorporated into the system to dynamically look for stack violations. This audit is performed by periodically checking the last byte of each stack and making sure it always contains a zero.

2.4 Time critical processes

Time critical processes must respond quickly to call processing

inputs. The ROB (Remote Order Buffer) process is the most general time critical process at the RT. It is the primary base level workhorse and is scheduled as a class A task. In the normal host-controlled mode, ROB processes receive data link orders from the ESS host, execute these orders, and return a success or fail acknowledgment. In the stand-alone mode, these processes take over the role of handling entire telephone calls themselves, one transient telephone call per ROB.

The process descriptor for a ROB process is quite large. The rp and sp stacks account for most of this size and are estimated to provide 10 levels of C function nesting. In addition, there is a third stack in each descriptor for bank switching. The top of this third stack is located by a 16-bit pointer in memory called the Bp (bank pointer). This stack is used by the trap handler to store the current bank number and return address when an interbank call occurs. Note that it is important for the trap handler to be reentrant since other interbank calls can occur during an interrupt or during a time break.

When a ROB process takes a time break, it calls an operating system subroutine and passes it a timing parameter. This subroutine saves the current values of the rp, sp, Bp, and bank number in the process descriptor, restores their previous values, sets up the specified timing, and returns to the operating system. Later, after the specified time has passed, these pointers are restored and control is given back to the ROB process via a subroutine return. This method of context switching is very efficient since it involves little more than saving and restoring several stack pointers.

Another important time critical process is the USC (Universal Service Circuit) process which is scheduled at the 10-ms interrupt level to ensure short and accurate timing. A USC is a hardware circuit with metallic access to a group of lines. It is primarily used to ring telephones but can also perform power cross tests, party tests, and coin control functions.

In contrast to ROB processes, USC processes are designed to perform a very specific task in a very efficient manner. For example, the USC process descriptor contains a small sp stack only large enough for a few return addresses. During execution, the USC process uses a common rp stack but any information left in this stack during a time break is lost. Timing during a time break is provided by linking the USC process descriptor at the proper point for the amount of delay requested to a timing list especially designed to minimize operating system overhead. Any delay in multiples of 10 ms up to 1 second can be requested.

The remaining time critical processes are controlled by the RT operating system on a so-called one-shot basis. No time break mechanism is provided for such processes; they are called periodically, perform some quick task, and return. Any change in per-entry behavior

of such processes occurs as the result of changes in information read from the periphery or from global data structures. These one-shot processes use the same common rp, sp, and Bp stacks as the operating system. A good example of such a process is the line-scanning process which is called frequently to look for potential line originations or disconnects. This process also performs hit timing and flash timing.

2.5 Time deferrable processes

The remaining processes are referred to as time-deferrable processes because they do not handle customer affecting inputs. For this reason, they are deferrable and, during heavy load, can be given control less frequently.

The RT operating system supports three kinds of time-deferrable processes: (*i*) audit processes, (*ii*) BGT (BackGround Task) processes, and (*iii*) a special data link diagnostic process. There is one process descriptor for each of these three kinds of time-deferrable processes and hence only one process of each kind can be active at a time. These processes are all normally scheduled as class C tasks but in some cases can enter higher priority modes of execution. One case occurs when a ROB process is blocked by a resource seized by a diagnostic BGT process. At this point, the BGT process begins executing as a class A task until the resource is released. This action minimizes the amount of time the ROB process is blocked.

The purpose of an audit process is to examine global data structures to verify that they contain legitimate information. This service is necessary in continuously running systems to free resources that otherwise get lost slowly as the result of program bugs or hardware failures. In the RT, only one audit process is active at a time and, when it completes, the next audit is automatically started in a continuous audit cycle. To support time breaks, the audit process descriptor contains three stacks which are identical in function and similar in size to the ROB process stacks.

Also identical in function but much larger in size are the stacks of the BGT process descriptor. This large size is needed to support the deeper subroutine nesting structure and greater use of local variables required by certain BGT processes. There are quite a number of different BGT processes to provide the various maintenance and administrative functions needed occasionally but not simultaneously at the RT. The bulk of these processes are diagnostics to help a craft person locate and repair hardware faults.

A BGT process is normally inactive for relatively long periods of time until a request is made for its service. If another BGT process is currently active, the operating system will queue a limited number of additional requests. The BGT processes are requested either automat-

ically on a timed basis or manually on a demand basis or in some cases both ways. The BGT processes are also unique in that they can be aborted manually by host TTY input message.

The BGT processes can also execute at interrupt level by calling a special operating system subroutine and passing it a delay parameter. This subroutine uses the delay parameter to initialize a programmable timer that causes an interrupt when it times out. When the interrupt occurs, control is given back to the BGT process via a subroutine return. This form of time break can be used any number of times in sequence, effectively allowing a BGT process to raise its priority of execution above base level. Normal base-level execution is resumed by taking a standard time break. This feature is valuable when diagnostics need to delay an accurate amount of time.

To the operating system, BGT processes appear as subroutines that are called and return when done. This structure also allows one BGT process to call other BGT processes as subroutines. This hierarchical structuring ability has the valuable consequence of allowing one to build a large process from several small ones. One important use of this feature is in the routine exercise BGT process run nightly at the RT. This process calls, in a prescribed sequence, all of the diagnostic processes which are otherwise requested manually on an individual basis.

The data link diagnostic process, which performs the necessary testing at the RT end of the data link, is under control of the host. This process has its own process descriptor so that the host may diagnose the data link any time it wishes without being blocked by other activity at the RT. The data link diagnostic process also has a unique interface with the operating system. It may call a special operating system subroutine that returns control on the next off-line data link interrupt. Repeated calling of this subroutine allows this process to change its mode of execution from base level to data link interrupt level, a feature essential to completely test the data link.

III. DATA LINK MESSAGE HANDLING

Messages sent over a 2400-b/s serial data link provide the primary means of processor-to-processor communication between the RT and the host. The link is a two-way communication path with each end capable of sending and receiving concurrently. For reliability, the data link is duplicated with the normal configuration being one link in the active mode and the other link in the standby mode. Both RT microprocessors have access to both data links which allows either microprocessor to be active, independent of the data link configuration. In addition, it allows the active microprocessor to communicate over both links concurrently for testing or other purposes.

The data link interface hardware at the RT converts a serial bit stream on the link to and from (8-bit) bytes of information for the RT firmware, plus performing certain control and checking functions. The RT data link message handling firmware is completely interrupt driven by the interface hardware. Each byte received causes an interrupt at a maximum rate of 3.3 ms per byte. Similarly, at the same maximum rate, an interrupt is caused each time the hardware is ready to transmit another byte. On the data link, a limited number of data bytes are grouped together, along with certain control and checking bytes to form a frame which is transmitted according to a standard protocol. The frame protocol is designed to provide a high degree of error control. Data bytes are initially sent and ultimately received in a higher level protocol termed a message. The message protocol is designed to provide a unit of information exchange between RT and host programs and simply contains message delimiting, routing, and size information, plus data. A message may be contained in any fraction of one or more frames.

3.1 Frame protocol

The frame protocol is the standard CCITT X.25 format of the Synchronous Data Link Control (SDLC) protocol. It is a positive acknowledgment/retransmission scheme with extensive checking to provide a high degree of error control for a single data link. Configuration and state control of the duplicated data links is handled at the message protocol level. The X.25 protocol has special commands to bring up or disconnect the link. There are also special commands to turn off and restart transmitting to prevent sending data at a rate faster than it can be received.

All traffic over the link, whether it be special commands, responses, or actual message data, is transmitted in the form of frames. A frame consists of from 6 to 22 bytes in the following fixed sequence:

1 flag byte,
1 address byte,
1 control byte,
0 to 16 data bytes,
2 CRC (cyclic redundancy check) bytes, and
1 flag byte.

An idle link is marked by a continuous sequence of ones. A flag byte consists of a zero followed by six ones and a zero, (01111110) and denotes the beginning and ending of a frame. A single flag may also close one frame and open the next frame. The address byte simply distinguishes between commands and responses. The control byte distinguishes the different frame subtypes and also contains the sequence number information for the positive acknowledgment proce-

dure. Data bytes have unrestricted value and the hardware automatically inserts and deletes a zero bit after all sequences of five contiguous ones to ensure a flag is not contained in the data. The CRC bytes are also generated and checked by hardware to detect any errors introduced during the transmission of the frame.

All information frames are numbered modulo 8. Up to seven information frames may be sent before transmission must stop to wait for an acknowledgment. Frames are acknowledged by sending the sequence number n of the next expected frame, thereby acknowledging all frames with sequence numbers n-1 or less (modulo 8). All unacknowledged information frames are retransmitted after an appropriate timeout period to take care of cases where information frames or acknowledgments are lost on the link. In retransmission cases, the sequence numbers per frame are very important in distinguishing new frames from retransmitted frames to prevent accepting the same frame more than once.

3.2 Message protocol

Messages to be sent to the host are placed in a circular transmit buffer where they are extracted one byte at a time by the interrupt driven data link control program and placed in the data section of an outgoing frame. Conversely, data bytes from incoming frames are placed one byte at a time at interrupt level into a circular receive buffer where they are partitioned at class A base level into individual messages. A message is a sequence of 16-bit words with the following simple format:

1 sync word,
1 message header word, and
0 to 32 data words.

The sync word is a fixed bit pattern and is used to find the beginning of a probable message after power up or any other cause of message mutilation. The message header contains the client identity and message size information needed to properly route messages. Data words have unrestricted value.

When a complete message has been received in the data link receive buffer, the entire message, without the sync word and usually without the message header word, is moved by the RT operating system into the specified client message buffer. Each client buffer has a load and unload pointer associated with it, and these pointers are updated properly to notify the client of a new message. Each client scans its buffer for new messages at a frequency dependent on load and priority. As orders contained within the message are executed, the client buffer pointers are adjusted accordingly until the two pointers are equal signifying an empty buffer.

3.3 Link state control

The data link transmit buffer and receive buffer, plus associated pointers and protocol state variables, are organized into a single structure termed a data link record. There are actually two data link records, one always associated with the on-line (or active) data link and the other always associated with the off-line data link. The RT keeps a single flag denoting which link, 0 or 1, is associated with the on-line record.

The two data link records give the RT the capability of receiving and transmitting messages over both links concurrently. However, messages to be transmitted are normally placed in the on-line record, unless a specific request is made for the message to be placed in the off-line record. This is done only for certain test and error messages.

Configuration of the duplicated data links is entirely the responsibility of the host which informs the RT of its decision via a special data link switch message. This message specifies which link, 0 or 1, is to be associated with the on-line record. Upon receipt of such a message, if the specified link is currently associated with the on-line record, nothing is done. If not, the RT flag is toggled and the on-line record is immediately associated with the opposite link. Any loss or duplication of data caused by this switch is cleaned up by the frame protocol.

The RT firmware has extensive data link error control and recovery checks designed into it. The data link records are audited frequently by normal message and frame handling programs for protocol violations, out-of-range pointers, and other illegal states. In addition, the data link records are audited once a second at base level for deadlock conditions. Protection against hardware failures is provided by masking off invalid or too-frequent data link interrupts to prevent them from overloading the microprocessor. Either link interrupt may be independently masked. Once a second at base level, an attempt is made to reinitialize the data link hardware associated with any masked interrupt and, if successful, the interrupt is again unmasked.

As the last line of defense in helping the host reconfigure and recover the data links properly under all possible situations, three special RT-to-host messages have been implemented. The first message informs the host that the RT has received an input message with improper message protocol. Upon frequent reception of this message from the RT, the host should assume a data link hardware failure has occurred. The second special message is sent by the RT when it finds an input message in the off-line data link record for a client that always receives messages from the active link. Since this evidence suggests that the host and RT no longer agree on the active link, this message is sent via the off-line record and asks the host to send a data link switch message specifying the currently active link. The last message is sort of a "heart

beat" since it is sent every 10 seconds by the RT. Each time the host receives this message, it returns a special acknowledgment message. The absence of this handsbake arrangement for a period of time indicates to both the host and the RT that one or both of the links are down. It is the responsibility of tbe host to initiate recovery actions in this case.

IV. METALLIC ACCESS CONTROL

Telephone switching systems exist to allow customers to talk or send data to each other. The voice and data signals are similar to each other, and are characterized by modest power levels and a frequency range of 200 to 3200 Hz. If the RT were required to carry these kinds of signals only, then the solid state voice network that it has would suffice. There is another set of signals, however, that is incompatible with this solid state network. These signals bave too much power (ringing), or have too low a frequency (party test), or both (coin collect) to be carried by the voice network.

An auxiliary network is necessary to apply these signals to customer lines. Since there is much less signaling than voice traffic, this auxiliary network can be more concentrated than the voice network.

In other switching systems, the high-power signals are provided by circuits called service circuits. These service circuits are usually single-purpose circuits, such as ringing circuits or coin control circuits. The No. 10A RSS RT design has a general-purpose circuit providing ringing, party test, and coin control. This circuit also provides dc access to the customer lines for other more specialized circuits. This general purpose circuit is called the Universal Service Circuit (USC).

The Line Interface (LI) boards have A and B metallic access relays per line circuit. Each metallic access relay connects a line circuit to a bus, and each bus is wired to relays of two USCs. (See Fig. 2.) Each pair of USC's has access to eight of these buses, and there are thirty-two LI circuits wired to each bus. Thus, each group of 256 lines has access to a group of four USCs. Note that the access network has the possibility of blocking.

4.1 Universal service circuit time sharing plan

Metallic access buses are used mostly for ringing. In North America, ringing usually consists of a two-second burst of ringing followed by four seconds of silence. Note that in the four seconds of silence, a single source of ringing current could be ringing two more phones by reconfiguring the access between the ringing current source and the phones being rung.

The No. 10A RSS takes advantage of this capability to reconfigure the metallic access network every two seconds. Thus, one USC can

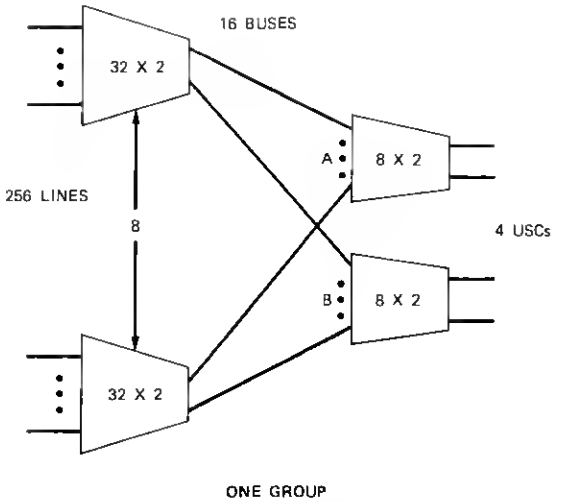


Fig. 2—Metallic access network.

supply ringing to three phones. The metallaic access control software is written to give each USC three time slots, and each time slot is two seconds long. In the case of normal ringing, a time slot is used to apply the 20-Hz ringing voltage to the line and scan for customer off-hook (ring trip). During the 4-second silent interval, no USC is connected to the line and the LI circuit is scanned for customer off-hook (ring trip). The beginning and end of each of these time slots is defined by a program entry. The program looks for work to do in a per-time slot database.

Each group of four USCs has two A metallic access bus USCs and two B metallic access bus USCs. The B USCs are delayed in time with respect to the A USCs by one-half time slot. Thus, any given line at any time has the potential of one complete time slot available starting in one second or less.

In a large RT, there may be eight groups of four USCs. The time slots corresponding to these eight groups are staggered in time. This is done because starting a time slot or ending a time slot requires many processor cycles. Spreading the group start-up periods in time spreads this processor load out in time.

There is a set of programs called the USC operating system which is driven by the 10-ms timed interrupt. These programs keep track of what time it is, start the control programs (which actually connect the USCs to the lines desired and accomplish the desired functions), provide real time breaks, and tell the USC selection routines what time it is so that the routine can select the next time slot.

4.2 Universal service circuit administration and timing

To understand how USCs are administered, it is necessary to briefly describe the USC administration databases.

There is a buffer per USC containing per-USC information and a small stack. The *BELLMAC-8* stores return addresses on the stack. Putting a stack in each USC buffer allows the control programs for each USC to take real-time breaks easily. All that is required is for the real-time break routine to remember what the value of the sp was when the break was taken, and to restore the sp after the break is over. The control routine may resume processing, since the string of return addresses necessary to get back to the first subroutine call is intact and pointed at by the sp.

Each USC buffer contains three time-slot buffers with information pertaining to the USC function being performed in that time slot, and the necessary pointers to exchange information with other programs.

Each USC can ring three different phones at any time by using the three time slots. Thus, there must be three busy/idle bits per USC for USC selection purposes.

To select a USC time slot, it is necessary to know

(*i*) The number of complete time slots necessary to hold the function to be performed. If only a partial time slot is desired, then the length of time (in tens of milliseconds) necessary to accomplish the desired function.

(*ii*) How far ahead in time the USC selection routine is to look. This prevents long delays, i.e., instead of selecting a time slot four seconds away to do a party test, the selection routine will return a blocked indication.

(*iii*) What time is it now with respect to the time slots of the USCs that can service the line in question. This information is necessary so that the routine can select the next occurring idle time slot.

After deciding which time slot or time slots can hold the function, the USC selection routine then makes those time slots busy by marking the time slot busy/idle bits.

The USCs are normally administered independently of the host. This is necessary because USC time slots must be chosen in time order, and it would be extremely difficult to keep the host and the RT in precise synchronism.

After the time slot is chosen and seized, the time slot buffer or buffers are loaded with the information necessary to do the function. If the time slot chosen is occurring at the time it was selected, the control routine is started immediately. If, however, the time slot is due to start at a later time, the time slot is started later by the USC operating system.

The USC control routines must report the status of the function

performed to the routine that initiated the function. If the function is regular ringing, then the report occurs only if the function was started immediately. This prevents long holding times on Remote Order Buffers (ROBs) on high-runner calls. Functions other than ringing always get reports from the USC control routines.

In the control of the USC, there are occasions when waits are desired. For example, when relays are operated there will be several milliseconds between the signal that puts current through the relay coil driver and the actual closing of the contacts. The USC operating system, as mentioned above, can suspend processing by remembering the value of the sp at the point where the real-time break is requested. The USC operating system has a series of linked lists for taking real-time breaks. Each list corresponds to a different real-time break, i.e., there is a list for 10-ms breaks, one for 20 milliseconds, etc. The USC operating system links the USC buffer onto a linked list when the control program calls the real-time break routine. After the time is up, processing resumes on that USC by restoring the sp (stored in the USC buffer) and resuming processing. This scheme was adopted because it can be administered very efficiently.

4.3 Universal service circuit functions

When ringing telephones, some of the tests made in the first cycle of ringing are the following:

(*i*) A test is made to verify that no other LI circuits are connected to the metallic access bus used for this particular USC-to-LI connection. This is necessary to avoid interference between the two circuits connected to the bus.

(*ii*) There are USC and LI logic tests done to assure that the circuitry used to ring the phone is in proper working order.

(*iii*) A measurement is made to assure that there are no high voltages on the outside plant. This measurement, called a power cross test, protects the USC used to ring the line.

Subsequent ringing cycles for the same call eliminate the above tests, except for the first metallic access bus test.

Reverting ringing is used to alert the calling customer and the called customer if both parties are on the same party line. This kind of ringing requires two time slots, one for each customer. Instead of writing a special control program for reverting ringing, this kind of ringing uses two time-slot buffers, each loaded with the information to ring one of the associated parties.

There are three basic kinds of coin orders: collect, return, and presence test. Coin collect and return orders put 130 volts onto the tip conductor. The difference between the two functions is the polarity of the voltage. Coin presence tests put a positive or negative potential on

the tip conductor and test for the presence or absence of a ground by whether or not current flows.

Party test is almost exactly like the coin presence test; that is, a potential is applied to the tip conductor and the presence or absence of current is tested using the USC measurement capability.

V. LINE SUPERVISION PROCESSING

In the 10A RSS, the host system has overall control of calls, but it does not have direct access to the line supervisory states. The only call-processing circuits in the RT that can scan the line are the USCs (during ringing) and the LI (all other times).

The RT controller has the responsibility of scanning RT lines and sending the supervision states to the host. There are two paths for this information: via the channels or via the data link. If a line is connected to a channel, the supervisory state of the line must be written to the channel because the solid-state voice network will not pass dc signals. The supervisory state that is written to the channels by the RT processor is reflected by the channel circuit at the host and used to supervise the line. The data link is used for reporting supervision changes for lines that are not connected to a channel.

5.1 Off-hook and on-hook scanning

The RT has an off-hook scan program and an on-hook scan program. Each program is run every 200 ms, and the two are run 100 ms apart. Not every line is looked at by each program. For example, lines that are in the origination (idle) state are not looked at by the on-hook program, since on-hook lines in the origination state are of no interest to the host. This selective scanning is done by having a mask table for each routine. Lines in the origination state have ones in the off-hook mask table, corresponding to their equipment locations, and zeros in the on-hook mask table.

When the off-hook or on-hook scanning routine finds a line in the state for which it is looking, the routine must consult an item called a Path Memory Remote (PMR) table entry to find out what to do with the line. For example, assume that the off-hook scan routine finds an off-hook and then consults a PMR. If the PMR indicates that the line is in the origination state, the off-hook is treated as an origination. If the PMR indicates that the line is connected to a channel, the off-hook is treated as a change of state and the channel supervisory flip-flop is updated.

The functions that can be performed by the off-hook and on-hook scan routines are as follows:

(*i*) The programs pass the supervisory state of lines that are connected to channels to the channels. This is done by scanning the

lines as they go into the pass supervision state and writing the appropriate state to the channel supervisory flipflop. Then a bit is set in the opposite scan mask table so that changes of state will be discovered by the opposite routine. When the routine discovers a change of state, it updates the channel circuit supervisory flip-flop, resets the bit in its own scan mask table corresponding to the line just found, and sets the bit in the opposite table.

(*ii*) The programs perform timing by setting both of the bits in the scan mask tables and using the PMR to count time intervals. Since both bits are set, the timer is incremented every 200 ms regardless of the supervisory state of the line.

(*iii*) The programs inform the host of line supervisory changes by scanning for a given change, then sending a message to the host when it is found.

(*iv*) The programs must scan intra-RT calls for disconnect, since intra-RT calls use no channels to the host. Since one or both of the parties may have flashing privileges, the RT may have to time the length of on-hook intervals to make sure that they are not flashes.

(*v*) The programs discover originations. This is discussed in the next section.

5.2 Origination processing

After a line originates, the origination is verified by doing a rescan and making sure that the line is not back on-hook. The line is then put into a buffer called the origination hopper and an origination message is sent to the host.

After the message has been sent to the host, the line is timed. If the host does not respond in a timely manner, the origination message is sent again. Another timeout causes a special origination message to be sent to the host which can clear up some RT and host data inconsistencies which can cause lines to be ignored.

Line Load Control (LLC) is an origination restricting feature that a switching machine can invoke when the machine is overloaded with traffic. If the host machine gets into this condition, it sends a message to the RT to go into the LLC state for three minutes. When in this state, the RT scans only certain high priority lines such as fire and police numbers and one-seventh of the remaining customers. The one-seventh of customers that are scanned is changed every time through the off-hook scan routine.

The RT drops out of the LLC state if the LLC message is not received at least once every three minutes.

The RT can recognize an overload condition within itself. This occurs when regularly scheduled work is not done, and results in the RT

suspending origination scanning for a brief period of time. This is independent of the LLC condition mentioned above.

5.3 Fast passing of supervision

For lines in certain states, supervision must be passed at a faster rate than allowed by the off-hook and on-hook scanning routines. This is done by interrupt level routines that pass supervision at 10- and 20-ms intervals.

To prevent this passing from becoming a burden to the processor, very special measures must be taken. The database described below makes this passing of supervision a tolerable burden.

The *BELLMAC*-8 chip has no on-chip registers where data are held and processed, but rather uses main memory for this data storage. The area of memory being used as registers is pointed at by the rp. Register-oriented commands are really memory manipulation commands.

Since the registers are really memory, it is possible to point the rp to memory that has been loaded with the data necessary to do some function. This simple operation has the effect of loading that data into registers.

In RSS, this scheme is used to save processor cycles for the fast passing of supervision. The data necessary to pass supervision is stored in memory in the form of linked lists, i.e., each piece of data that is used to pass supervision on one call has the address of the next piece of data. Loading this address into the rp has the same effect as mentioned: the data are loaded into registers.

There are two cases where this fast supervision is necessary. When a dial pulse line is dialing, a dial pulse can shrink to as low as 11 ms in length. Thus, lines in the dialing state need passing of supervision at a 10-ms rate.

Lines that are being rung also need fast passing of supervision so that the ring trip at the RSS is processed promptly at the host. This need is not so stringent, so passing of supervision on lines being rung is done at a 20-ms rate.

The ringing pass supervision routine also must do some processing on lines that have tripped ringing. The ringing connection that is up (if the ring trip occurred in the two-second active interval of the ringing cycle) must be torn down and the USC and metallic access bus made idle. The channel must be put into the talk state, and the LI must be put into the talk state.

VI. REMOTE ORDER PROCESSING

Electronic switching systems are generally designed as call-oriented systems. A call coming into the system is assigned a memory block,

and control of the call is passed through various programs to handle particular parts of the call. However, the primary focus of the system is still the call.

The RT of the No. 10A RSS is not call oriented. Instead the central focus of the RT is the processing of remote orders from the host. The RT orders consist of operations to be performed on the RT peripheral hardware. Each order describes both an operation and a logical circuit on which to perform it. Examples include path orders, line scan orders, ring orders, and coin orders. These orders which correspond to particular parts of a call are under control of the RT, but total call control is resident in the host.

The philosophy of this control between the host and the RT is one of a master/slave relationship. The host, being the master, controls the actions of the RT on a functional basis. Orders to execute specific functions are sent to the RT over a data link. The RT, being the slave, receives these orders and executes the functions. Acknowledgments indicating the status of order execution are returned to the host.

The master/slave relationship provides a simplified view of the operation of the host and RT. The orders received from the host over the data link can cause quite different actions to be taken by the RT, depending on the state of the lines involved in the order. For example, an order to change the supervision of a line causes different hardware and firmware actions for a line in an intra-RT connection than for a line in an RSS to ESS connection. The RT responds to all data link orders as an intelligent slave and bases its actions on the current line states.

6.1 Remote order buffers

The ROBs are used for sending call-processing orders from the host to the RT. The host call-processing programs will seize an ROB and load it with one or more orders necessary to process a part of a particular call. The contents of this ROB will then be sent over the data link and stored in a corresponding ROB in the RT. Upon completion of the ROB orders, the RT returns acknowledgment messages over the data link to the host ROB. Programs in the host continue processing the call based on the acknowledgment data.

The host has a pool of ROBs used for sending orders to the individual RTs. The ROBs are not dedicated to a particular RT; they can be used to send orders to any RT. However, the number of ROB identifiers available for each RT is limited to the number of ROBs present in the RT. This prevents blockage of ROB data in individual RTs. The average holding time for an ROB is approximately 1 second. The ROB execution within the RT contributes most heavily to this holding time.

6.2 Data structures

Several data structures are required for remote order processing in addition to ROBs. Since the RT is a slave to the host, the primary data structures for call control are resident in the host. Call registers, resource memory, and equipment translators are all maintained in the host for ready access by call control. Data structures that are maintained in the RT are used primarily for maintenance and stand-alone.

Resource memory for the lines, channels, and network are kept in the host and duplicated in the RT. The state of the lines and channels is stored in PMRs and Path Memory Channel words (PMCs), respectively. Network path information is reflected in Path Memory Junctor words (PMJs) and the network map. Busy/idle status of network links is stored in the network map, while path descriptions are stored in the PMJs.

During the execution of remote orders, the RT memory for these resources is kept up to date. This allows the RT maintenance programs, which are independent of the host, to select and use idle resources. During the time the maintenance programs are using resources, the appropriate resource memory is marked diagnostic busy. The only conflict arises if the host chooses to use the same resource that maintenance is using. In this case, the remote order notes the resource conflict and requests maintenance to idle the desired resource. After a time break to allow maintenance to idle the resource, the remote order proceeds. This is an example of the master/slave relationship between the host and the RT.

The resource memory is also necessary for the entry and exit transitions from stand-alone operation. Intra-RT stable calls are maintained during the transitions through preservation of the resource memory. (See Section VII for details.)

Remote order processing also makes use of the resource memory for the metallic access network. However, full administration of this network exists in the RT so that duplicate memory and resource conflicts with the host do not exist. (See Section IV for details.)

The RT equipment translators exist primarily in the host for ready access by call control when forming remote orders. However, a few translators are also maintained at the RT to provide information required during remote order execution. These translators include:

(*i*) Line Remote Equipment Number Translator provides line transmission and ground start information for lines.

(*ii*) Ground Start Applique Translator relates lines to ground start applique circuits.

(*iii*) Remote Miscellaneous Distributor Number Translator provides distributor point information for lines.

(*iv*) Office Parameters are used to specify variable information for the RT such as ringing type and equipage of circuits.

6.3 Remote order buffer execution

Once a ROB has been loaded with orders from the data link, the RT operating system schedules execution of the ROB. The ROB execution program, which deals with the data in the ROB, can be viewed as a three-level structured program. The highest level in the structure is the executive which receives control for an active ROB from the operating system, decodes orders in the ROB, and passes control to the order execution routines. The executive also handles the acknowledgments returned to the host after the ROB orders are completed. The second level is a series of programs which carry out the order execution requested by the executive. To perform the actions requested by the orders, these routines call a third level of programs which are collections of various primitive functions that carry out specific tasks.

The operating system regularly looks for active ROBs that need to begin or continue execution. When a ROB with pending orders is found, the ROB executive begins processing by extracting the first order identifier and decoding it. Assuming a valid order is found, the executive reads the remaining data in the ROB associated with the order and passes it on to the appropriate order execution routine. Upon completion of the order, the order execution routine returns success/fail data to the executive. If the host requested acknowledgment data after the completed order, the executive returns the success/fail data to the host ROB via the data link. The executive then extracts the next order identifier in the remote ROB and repeats the process. The ROB processing continues until all orders in the ROB have been completed.

If an order should fail execution for any reason, the executive will report the failure to the host via an acknowledgment. If the acknowledgment was requested after the failing order, an immediate report is sent. However, if another order follows the failing order, that order and all succeeding orders will be skipped until an acknowledgment request is found. At that point, the failing order data are returned in an acknowledgment to the host and ROB execution terminates.

Another option that can be requested by the host for remote ROB execution is to ignore all failures during order execution. This option is used during failure recovery when the host program idles the RT hardware. Since the state of the hardware may be unknown, failures on some orders are expected. The failures are purposely ignored by always returning success acknowledgments to the host. This option simplifies the host program by eliminating the processing of ROB failure conditions.

After the executive has decoded an order identifier and extracted the data for the order, control is passed to one of many order execution routines. These routines break down the orders into a sequence of primitive functions dealing with specific tasks in firmware and hardware.

For example, the "path" order contains data specifying a set of two Remote Equipment Numbers (RENs), two network A-links, and a junctor which describe a unique network path. Additional data specify a connect or a disconnect operation and the new scan states for the two RENs. To execute this order, the path order execution routine must first translate the logical equipment numbers into peripheral hardware addresses. These addresses are in turn used in specific tasks to perform the network hardware operations. The new status of the network is recorded in the network maps and the PMJs. Similar tasks must be performed for each of the line scan states. Each one of the above functions must be completed to perform the path order. A failure in any function causes the order to fail and a failure return to the ROB executive.

Each ROB order is made up of one or more primitive functions which perform a specific task in either hardware or firmware. Several functions are used in the execution of the path order described above. One of these functions records the new status of the network in the data structures. This is accomplished by writing the path data into the PMJ for the junctor, computing the network links, and writing their new state into the network maps. This is an example of a function that depends on no other condition than the path data passed to it, and the function cannot fail.

Other primitive functions depend on the current state of the hardware. For example, setting a line to the origination scan state causes different actions when the line is in a path to the host than when it is in a path to another RSS line. In the first case, the line's supervisory state is repeated over a channel to the host. In the second case, the line's supervisory state is sent to the host as a data link supervision report. In either case, when the line disconnects, a ROB order will be received at the RT to set the line to the origination scan state. Clearly, the primitive function which handles the scan state change must first determine the old state of the line. The appropriate set of actions can then be taken to realize the desired final scan state. A success/fail return is given to the higher level routine to reflect execution status of the function. In this example, the primitive function is dependent on the line state and may fail.

Many of these primitive functions are called by several of the order execution routines. By placing the line state dependent actions in the primitive functions, the order execution routines do not require knowl-

edge of previous line states. This achieves greater code efficiency and more uniformity in handling specific tasks.

6.4 Peripheral control

Many of the primitive functions called by order execution routines perform peripheral hardware actions. However, not all peripheral circuits operate in the same amount of time. The electronic PNPN network operates at program speed and requires no firmware time breaks. Other circuits, such as the metallic access network, take longer to operate and require the firmware to delay before proceeding with additional orders. Consequently, a provision for time breaks has been included in the primitive functions to handle the slower operation. Higher level order execution routines are unaware of these delays.

Also built into the peripheral control functions is the ability to retry orders. If an order fails, it is first retried on the same microprocessor controller. If the order fails a second time, the microprocessor controllers are switched and the order is retried from the newly on-line controller. Only if this third attempt fails is the peripheral circuit considered faulty and the order aborted. This retry strategy gives the peripheral orders every chance to succeed before a failure is accepted and propagated back to the host through a ROB acknowledgment.

VII. STAND-ALONE SERVICE

The stand-alone feature of the No. 10A RSS RT provides a POTS-type intraoffice service to individual, multiparty, PBX, and coin-type lines. Typically, an RT will be located in an area where there is a community of interest. The feature has been designed to serve this community of interest during periods when both data links to the host fail.

The feature is a Bell operating company (BOC) option and is implemented by equipping the RT with an engineered quantity of special circuit boards which provide the tone digit receivers and tone sources. The stand-alone feature is always resident in the RT firmware.

7.1 Stand-alone features

The stand-alone feature was designed primarily with the CDO application in mind. It provides intraoffice services to lines which are served directly by the RT.

Local emergency service agencies in a community (e.g., fire, police, hospital, etc.) will most likely have several local telephone lines working in a multiline hunt arrangement. To continue to provide an adequate service to these agencies, a basic multiline hunt capability has been included in the stand-alone feature.

A special facility has been incorporated to reroute selected numbers such as "O" operator and "911" to a local line. This rerouting is

accomplished by the Special Directory Number Translator which can translate an "O" first digit, or any 3-digit or 7-digit number to a directory number served by the RT. If the BOC provides an entry in the special number translator to handle "O" calls, the RT can handle manual lines in the stand-alone mode by routing manual line call originations to the special "O" handling.

Coin phones may be either dial-tone-first or coin-first. Intra-RT calls will be completed but any coins deposited will be returned. For multiparty lines, all ringing options are available.

A recorded announcement port has been included on the special stand-alone circuits boards. Provision of a recorded announcement machine is optional for the BOC. Calls destined to numbers not served by the particular RT are routed to the recorded announcement on a "barge-in" basis. If one is not provided, reorder tone is substituted for the recorded announcement.

While running in the stand-alone operating mode, no connect timing is performed on calls and no record is made for charging purposes.

Special features such as call forwarding, abbreviated dialing, call waiting, specialized multiline hunt patterns, etc., are not available.

Counts of traffic activity and of calls which cannot be handled for lack of or malfunctioning of resources are maintained. These counts are sent to the host and displayed on teletypewriters (TTYs) at the time the No. 10A RSS exits from the stand-alone mode.

7.2 Entering and exiting stand-alone

The stand-alone operating mode is entered whenever a duplex data link failure occurs. This may occur as a result of the loss of frame protocol on both data links for a period of 30 seconds or as the result of the loss of the "heart-beat" for a period of about one minute. A heart-beat message on the active data link is the ultimate back-up which controls entry into stand-alone. The RT originates a data link message every 10 seconds and increments a heart-beat counter. The host on receiving this message returns an acknowledgment which, when received at the RT, causes the counter to be zeroed. If this counter gets past 6, (no heart-beat returned for about 1 minute), all call processing in the RT is stopped and the transition into stand-alone is initiated.

When entering stand-alone, all stable intra-RT calls are retained. All other calls are disconnected and the affected lines are placed into the appropriate state (origination or high and wet). Placing off-hook lines into the high and wet state helps to control the number of originations entering the system when stand-alone processing begins.

The entry into stand-alone generates a modified transient clear

which initializes the database and hardware of all transient calls and all calls involving a channel connection to the host. It is possible that the trigger to enter the stand-alone mode may not affect the RT to host channel calls. However, since the stand-alone service circuits share network appearances with the channels, it is essential that channel hardware be idled.

In the unlikely event stand-alone was triggered by other than carrier system failure, the channel idling will ensure that the host ESS will clear all of the related RSS calls. In the event of a normal outage, the host ESS is protected by the Carrier Group Alarm (CGA) conditioning capability of the host channel circuits.

The process to bring the RT out of the stand-alone operating mode is initiated by the host. This may be accomplished manually by the initialization of the data links at the host or automatically under control of host code. The exiting stand-alone process is started by a host-originated message in which the host asks the RT "Are you in Stand-Alone?" If the RT is not in stand-alone, it sends a reply which so advises the host and does nothing.

If the RT is in stand-alone, all call-processing work is suspended and the exiting stand-alone process is executed. As in the case of entering stand-alone, a transient clear is done which again initializes the database and hardware of all transient calls. All lines which are off-hook, including both those involved in transient calls and those giving a permanent signal (high and wet lines) are placed into the originating state. Stable calls remain connected.

When the database and hardware initialization work has been completed, information on stable calls is sent to the host. This information is used by the host to rebuild its database.

Finally, the RT sets up a background task to send the traffic data, which was collected while in stand-alone, to the host for printing on the host's TTY. A message is then sent to the host that says "I am out of stand-alone" and normal host-RT call processing resumes.

In the RT exiting process, lines with permanent signals are placed in the originating state. The host, after resuming control of call processing, will see these lines as call originations. With no activity on the lines, they will be given permanent signal treatment by the host and will eventually be placed into the high and wet state at the host.

7.3 Data structures

During stand-alone call processing the Stand-Alone Buffer (SAB) is the basic data structure and serves as a repository for data which are accumulated while originating or disconnecting a call. Each call origination and each disconnect requires the use of an SAB. On an originat-

ing call, the SAB is idled when the called party answers or if the calling party abandons the call. In many respects its use is similar to that of a call register in other ESS systems.

The ROBS, which are used during host mode operations, are used and redefined as SABs for stand-alone mode. Because of much longer holding times (an average of 30 seconds for an originating call) additional SABs are provided for use only in stand-alone.

While a call is in the dialing stage, a Universal Call Register (UCR) is attached to the SAB and placed on the digit collection linked list. This is a reuse of the UCR from one of its normal host mode functions.

In the RT, office translation information is stored in write-protected memory. Principal translation data structures are as follows:

(*i*) Directory Number (DN) Translator provides the information to translate a called number to the equipment location where the called party's line terminates. This translator also provides ringing information to ring the called party's line.

(*ii*) Special Directory Number (SDN) Translator is used to translate the called special emergency numbers to a DN served by the RT. A first digit of 0, or any 3- or 7-digit number can be entered into this translator (e.g., "0" or "911").

(*iii*) Line Remote Equipment Number (LREN) Translator provides information on the attributes of lines: i.e., major class of service, ground start applique usage, sleeve lead, *TOUCH-TONE** service equipped, member of a multiline hunt group, etc.

(*iv*) Multiline Hunt List contains the LREN number for lines which are members of multiline hunt groups. This list is used in connection with the multiline hunt facility.

(*v*) Remote Miscellaneous Distributor Number (RMDN) translator provides translation information for lines equipped with sleeve leads.

(*vi*) Ground Start Applique (GSA) Translator provides information on GSA equipment assigned to lines.

(*vii*) Office Parameters provide information on office equipment availability, office options, etc.

The LREN translator, GSA translator, and office parameters are used in both the stand-alone and host operating modes. The other translation data structures are used only while in the stand-alone mode.

A full translation update is made automatically by the host over the data link on a scheduled basis, or a special update can be requested manually at any time. To accommodate service order activity, recent changes entered at the host will be updated individually almost immediately at the RT.

Busy/idle maps for the network junctors and A-links are maintained

* Registered service mark of AT&T.

in memory. In stand-alone mode, the maps are used to make path assignments to connect a tone to a line or to make a talking path connection. Another busy/idle map in memory maintains the status of stand-alone tone circuit assignments.

7.4 Call setup

A major objective in the design of the stand-alone call processing feature was the maximum reuse of code and functions which were developed for host mode operations. Many ROB execution functions were used "as is," except for a few that required some minor modifications to accommodate both operating modes. Line scanning and ringing functions were also reused along with many hardware control primitives.

Stand-alone call processing is carried out by a number of processes interacting with one another asynchronously and communicating through commonly accessed message areas located in the SAB.

As in the host mode, the Line Scanning process regularly scans the line interfaces for changes in the on-hook/off-hook state of the subscriber lines. When a call origination has been detected, the originating line is posted in the Origination Hopper. The Origination Hopper Unloading function, running at base level, unloads the hopper and, instead of sending a data link message, passes the LREN of the originating line to the directly coupled Originations Interface Routine. The Originations Interface seizes an SAB and spawns a process to handle the originating call.

The RT operating system regularly scans the SABs for newly spawned processes and, when one is found, the Stand-Alone Executive is called. The Executive controls the call through three main phases of the call setup process. In the first phase, the Originating Call Process Initialization, the SAB is initialized, a UCR is attached, the call is placed on the digit collection list, and the originating customer is given dial tone.

The Get Dialed Digits Process, the second phase, runs at base level and works asynchronously with the 10-ms interrupt level Digit Collection Process to receive the dialed digits from the customer, to analyze them, and to determine the equipment location of the called number. Digit analysis is performed after the first, third, and seventh dialed digits. On completion of the second phase, the called line has been checked for busy, line supervision is turned over to line scanning, the call is removed from the digit collection list, and the called line is ready to be rung. The Get Dialed Digits Process, while waiting for receipt of digits from the customer, regularly takes real-time breaks and returns control to the operating system.

The third phase, the Ring Called Line Process, builds the ring order and calls the ROB execution function which handles ring orders. This

function spawns a new process which will actually control the ringing, and the detection of answer on the called line. When the called party answers, the ringing control process posts a flag in the SAB and terminates. The Ring Called Line Process then establishes the talking path between the two customers.

This call is now in a stable state, line supervision has been turned over to line scanning, and the call is completed. The SAB is idled and the Call Setup Process terminates.

The mechanisms for detecting and handling hardware problems in the host operating mode are still active when running in the stand-alone mode. However, with the data links out of service and no way of reporting the problems, these mechanisms are of limited value. Heavy reliance is placed on the hardware and software audits to keep as much of the RT hardware available for use as practical.

To minimize the effect of a defective tone circuit on a stand-alone circuit board the various tones are assigned on a rotating basis. To avoid indefinite tie-up of any one hardware or software resource, time-out facilities have been incorporated throughout the Call Setup Process. If the preset time limits are exceeded, the customer is given appropriate tone treatment and the resource is idled.

7.5 Call disconnects

While a call is in the talking state, line scanning watches both lines for an on-hook condition. When one of the lines goes on-hook and so remains for a period of 1 second (the on-hook is not a hit or a flash), line scanning passes the information to the directly coupled Supervision Interface Routine. The Interface then searches path memory to locate the other party, seizes an SAB, and spawns a Disconnect Process. No distinction is made between the calling and called parties in processing a disconnect.

Line scanning can detect an on-hook from a reverting call disconnect or from a high and wet line going on-hook. In either of these situations, there would not be a second line and a search of path memory would fail. In these cases, the Supervision Interface will still spawn the disconnect process which will perform a "single line disconnect."

The operating system, as in the case of an originating call, finds a newly spawned process and calls the Stand-Alone Executive. The Executive determines that the SAB is for a call disconnect rather than a call origination and then passes control to the Call Disconnect Process.

On receiving control, the Call Disconnect Process first disconnects and idles any network path. For the first or only line involved, any coins which may possibly be still in the coin chute of a coin phone are returned, and the line is conditioned so that it can originate a new call.

If a second line is involved (a line-to-line connection), possible coins are returned on coin lines. A direct check of the second line's line interface is then made to determine its on-hook/off-hook status. If it is on-hook, the line is set up so that it can originate a new call. Should it still be off-hook, the line is placed into the high and wet state.

This completes processing of the call disconnect, the SAB is idled, control is returned to the Stand-Alone Executive and then to the operating system. The disconnection process then terminates.

When the second line which possibly was placed high and wet later goes on-hook, another disconnect process will be spawned. The Call Disconnect Process will then be concerned with only a single line and appropriate action will restore the line to the originating state.

VIII. SUMMARY

There is a growing trend to incorporate distributed processing in telephone switching systems. The No. 10A RSS is in the forefront of this trend and is part of a system consisting of a host ESS connected to one or more remote terminals (RTs). The host ESS and its associated RTs have a master/slave relationship via direct processor-to-processor communication over a data link. The RT is effectively an extension of the host ESS allowing it to now provide service for customers many miles away. This arrangement can increase substantially the range and utilization of an existing host ESS.

The RT processing power is supplied by the *BELLMAC*-8 microprocessor. One unique architectural feature of this microprocessor is the absence of on-chip general registers. All such temporary variables are located on two stacks in memory referenced by the rp and sp. This feature makes the context switching required when calling subroutines, handling interrupts, or switching processes very efficient since no data movement is necessary other than changing two pointers. These kinds of context switching occur very frequently in the RT firmware design.

Most of the RT firmware is coded in the high-level language C. The RT tasks performed are controlled by a process-oriented operating system in which each process descriptor contains its own stacks. The dedicated stacks keep time breaks from having a detrimental effect on the program structure of a process. The number of bytes of RT firmware code exceeded the 16-bit addressing capability of the *BELLMAC*-8 microprocessor so a memory bank switching scheme was incorporated into the RT design. Bank switching was hidden from process execution by traps to the operating system supplied by the overlay link-editor.

Ringing and other high-power signals are supplied in the RT from a general-purpose circuit termed a Universal Service Circuit (USC). A sizable portion of the RT firmware is devoted to the complex control

required by these USCs and their metallic access to customer lines. The most notable aspect of this USC control firmware is that one USC can be used to ring up to three phones via a time-sharing arrangement.

The RT firmware is responsible for periodically scanning lines and reporting any change in supervisory state to the host. If the line is connected to a channel, the supervisory state must be written to the channel because the solid state voice network will not pass dc signals. This has to be done as often as every 10 ms, while the line is in the dialing state, to properly pass dial pulses. If the line is not connected to a channel, supervisory information is passed over the data link. In these cases, the RT also performs any hit timing or flash timing necessary.

The RT is not call-oriented since total call control resides in the host. Instead, the RT focuses on executing orders sent by the host over the data link in ROBs. Several ROB execution processes can be active at any one time. If data link communication with the host fails, however, the RT enters a call oriented stand-alone mode of operation to provide customers with a POTS-type intra-RT service. In this mode, the ROBs are redefined as SABs and increased in number to support the increased holding time of their new call-oriented usage. One important aspect of the transitions into and out of stand-alone is that they have been designed to preserve all stable intra-RT calls.

IX. ACKNOWLEDGMENTS

The design and implementation of the No. 10A RSS remote terminal firmware was made possible by the contributions of many people. Several people who deserve special mention are J. M. Brown, R. W. Gunderson, M. K. Pieper, K. A. Radtke, R. R. Rundquist, S. R. Staak, and J. B. Truesdale.